D5.3 - Third ICOS Release: Complete ICOS version

Document Identification			
Status	Final	**Due Date**	30/04/2025
Version	1.0	**Submission Date**	30/04/2025

Related WP	WP5	Document Reference	D5.3
Related Deliverable(s)	D5.1, D5.2, D2.2, D2.3, D2.4, D4.2	Dissemination Level (*)	PU
Lead Participant	NKUA	Lead Author	Anastasios Giannopoulos Menelaos Zetas
Contributors	ENG, TUBS, NCSRD, ATOS, UPC, BSC, CEADAR, ZSCALE, XLAB, SIXSQ, IBM	Reviewers	Alex Barcelo (BSC) Konstantin Skaburskas (SIXSQ)

Keywords:
System Integration, CI/CD, Final Release, Testing, Integration, Assessment, Documentation

This document is issued within the frame and for the purpose of the ICOS project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101070177. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

(*) Dissemination level: **(PU)** Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page). **(SEN)** Sensitive, limited under the conditions of the Grant Agreement. **(Classified EU-R)** EU RESTRICTED under the Commission Decision No2015/444. **(Classified EU-C)** EU CONFIDENTIAL under the Commission Decision No2015/444. **(Classified EU-S)** EU SECRET under the Commission Decision No2015/444.

Document Information

List of Contributors	
Name	Partner
Anastasios Giannopoulos	NKUA
Menelaos Zetas	NKUA
Anastasios Kaltakis	NKUA
Gabriele Giammatteo	ENG
Maria Antonietta Di Girolamo	ENG
Marc Michalke	TUBS
Zied Jenhani	TUBS
Admela Jukan	TUBS
Sebastian Cajas Ordoñez	CeADAR
Jaydeep Samanta	CeADAR
Andres L. Suarez-Cetrulo	CeADAR
Ricardo Simon Carbajo	CeADAR
Kalman Meth	IBM
Andreas Oikonomakis	NCSRD
Jordi Garcia	UPC
Marisa Zaragoza	UPC
Andreu Catala	UPC
Hrvoje Ratkajec	XLAB
Ivan Paez	ZSCALE
Alberto Llamedo	ATOS

Document History			
Version	Date	Change editors	Changes
0.1	12/02/2025	NKUA, ENG	ToC
0.2	20/03/2025	NKUA	First draft of introduction for each section
0.3	04/04/2025	NKUA	More content in Section 4.1 and Appendix 2 (Integration tests reporting)
0.4	08/04/2025	NKUA, CeADAR, XLAB, UPC, IBM, TUBS	More contributions on new features of ICOS final release, Unit Testing reports, ClusterLink and Annex 3.
0.5	09/04/2025	ENG, NCSRD	Contributions to Sections 2, 3 and 4. Policy Manager Testing and Section 4.4. were also added.
0.6	14/04/2025	NKUA	More contributions to Annex 1 and 2, global document formatting, additions Section 4
0.7	17/04/2025	NKUA	Clean version out for internal review
0.8	26/04/2025	NKUA, BSC, SIXSQ	Clean version upon addressing internal reviewers' comments
1.0	30/04/2025	ATOS	Quality check and Final Submission

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Anastasios Giannopoulos (NKUA)	29/04/2025
Quality manager	Carmen San Román (ATOS)	30/04/2025
Project Coordinator	Francesco D'Andria (ATOS)	30/04/2025

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AI	Artificial Intelligence
API	Application Programmatic Interface
CI/CD	Continuous Integration / Continuous Deployment
CLI	Command Line Interface
CNI	Container Networking Interface
CVE	Common Vulnerability and Exposure
D&PE	Distributed and Parallel Execution
Dx.y	Deliverable number y belonging to WP x
FL	Federated Learning
GUI	Graphical User Interface
HTML	HyperText Markup Language
IaC	Infrastructure As a Code
ID	Identifier
IT-x	Project's Iteration x
LSTM	Long-Short Term Memory Network
ML	Machine Learning
OCM	Open Cluster Management
OS	Operating System
UI	User Interface
UMA	User-Managed Access
URL	Uniform Resource Locator
UTx	Unit Test x
UT-Fx	Unit Test x for Fetcher
UT-Lx	Unit Test x for LSTM
UT-Px	Unit Test x for Processor
WPx	Work Package x

Executive Summary

This document, titled D5.3 – Third ICOS Release: Complete ICOS Version, marks the culmination of the ICOS software lifecycle, presenting the final integration, testing, and validation outcomes of the ICOS Meta-Operating System (MetaOS). Building upon the progress reported in the ICOS Alpha (D5.1) and ICOS Beta (D5.2) releases, this deliverable consolidates and finalizes the developments across all ICOS software layers, providing a mature, stable, and production-ready platform.

The release incorporates substantial updates across unit, integration, and system-level components. These include the introduction of new features in the Meta-Kernel Layer, the Intelligence Layer, and the Security and Data Management layers, alongside the refinement of CI/CD workflows, API enhancements, Helm charts, and testing strategies. Notably, new components such as the ClusterLink, Policy Manager, Matchmaker, Metrics Export API, and Federated Learning modules were extensively validated and integrated, significantly boosting the system's intelligence and observability.

This document also outlines the systematic approach to software validation, showcasing expanded unit testing coverage, per-component Helm testing, and the first full-scale integration testing campaign. A dedicated testbed infrastructure was established, and realistic testing scenarios were scripted via the ICOS Shell CLI. These scenarios simulate user-facing operations including authentication, controller registration, job orchestration, and resource cleanup, and are supported by CI-validated test results. Additional tests performed under WP3 (reported in D3.3), such as the ClusterLink integration and Zenoh-based topology update propagation, have been incorporated to confirm inter-cluster communication and runtime adaptability.

The documentation also highlights the finalization of the ICOS Suites, enabling the deployment of ICOS Controllers, Agents, and Workers through standardized Helm-based packages. An open and public GitHub repository (https://github.com/icos-project) now hosts the source code, accompanied by extensive technical documentation published via the ICOS Developer Hub.

In summary, this deliverable not only concludes the integration and validation activities of the ICOS project but also presents a coherent, extensible, and well-documented platform, ready for uptake by external adopters, industrial stakeholders, and open-source contributors. Through close collaboration among project partners and iterative feedback from prior releases, the ICOS MetaOS has achieved full system maturity, and this document serves as its definitive release record.

1 Introduction

1.1 Purpose of the document

This document is provided alongside the ICOS Third release and aims to offer comprehensive information about the software components included in the ICOS final release. It outlines the structure of the release, the delivered functionalities, instructions for accessing the source code and software packages, as well as guidelines on deploying and using the software. As the culmination of the ICOS development cycle, this deliverable builds upon the ICOS Alpha and ICOS Beta releases, integrating feedback, improvements, and new functionalities developed since the last iteration.

It also describes the integration process and tools used during the development of the ICOS final version. A brief overview of the main integration activities, previously covered in the deliverables *"D5.1 - First ICOS Release: ICOS Alpha"*[1] and *"D5.2 - Second ICOS Release"*[2], is included. The release has been delivered at project's month 32, ten months after the second release ICOS Beta.

Furthermore, the document details the testing approach used to evaluate the quality of the ICOS final release. While the Alpha and Beta versions employed an end-to-end testing strategy and a per-component testing plan, this release presents the advancements in unit testing, integration testing, and the newly established testbeds that ensure seamless system-wide validation. This started with defining specific testing goals to identify the software aspects requiring assessment. A structured methodology was then applied to guide the creation of test cases, the selection of tools, the execution of tests, and the analysis of results. Each phase of this process, along with the test outcomes, is documented.

Lastly, the document reports all the integration efforts, particularly focusing on the different integration tests adopted during the final release. Key functionalities and integration test results are also reported.

This document is mainly targeted at technical teams—both within and outside the ICOS project—who are interested either in deploying and operating an ICOS system (and need to know how to access the software and documentation) or in contributing to the development of ICOS software (and need insights into the processes for integration, testing, and release). Thus, this deliverable may serve as a reference for developers, system administrators, and stakeholders, offering insights into the finalized software architecture, deployment procedures, and validation results.

In a nutshell, the primary objectives of this document are:

- To provide a comprehensive overview of the final release of the ICOS software, highlighting the latest features and improvements.
- To describe the integration and testing processes applied to ensure a stable and high-quality release.
- To document the methodologies, tools, and infrastructure employed in the Continuous Integration/Continuous Delivery (CI/CD) and testing workflows.
- To present the advancements in unit testing, integration testing, and the newly established testbeds that ensure seamless system-wide validation.
- To outline the plan for post-release support, dissemination, and future improvements beyond the project lifecycle.

1.2 Relation to other project work

The ICOS Third Release is an outcome of integration efforts taken in Work Package 5 (WP5) and is a direct continuation of the efforts documented in previous deliverables, particularly:

- D5.1 - ICOS Alpha Release [1]: Introduced the initial version of ICOS, laying the foundation for the architecture and core functionalities.
- D5.2 - ICOS Beta Release [2]: Expanded upon the Alpha release, incorporating improved integration mechanisms, extended functionalities, and enhanced testing methodologies.

Additionally, this deliverable aligns with key findings and system architecture elements documented in:

- D2.3 - ICOS ecosystem: Technologies, requirements, and state of the art (IT-2) [3].
- D2.4 - ICOS architectural design (IT-2) [4].
- D3.3 - Meta-Kernel Layer Module Integrated (IT-2) [5].
- D4.2 - Data management, Intelligence and Security Layers (IT-2) [6].

By leveraging the insights and developments from these deliverables, D5.3 finalizes the implementation, integration, and validation of the ICOS software system.

1.3 Structure of the document

This document is structured into the following sections:

1. **"Introduction"** – Provides an overview of the document's purpose, its relation to previous deliverables, and a summary of its structure.
2. **"ICOS Final Release Notes"** – Details the final updates, including a summary of new functionalities, enhancements in system integration, and modifications to the software architecture.
3. **"Release Integration"** – Describes the technical and procedural aspects of integrating the final release, including the tools, infrastructure, and methodologies employed.
4. **"Release Testing"** – Outlines the testing strategy used to validate the ICOS system, covering unit tests, integration tests, and the new testbed infrastructure.
5. **"Final Release"** – Summarizes the completed development cycle and discusses what was achieved in relation to the planned activities in D5.2.
6. **"Conclusions"** – Highlights key lessons learned, challenges addressed, and the future outlook, including post-release support and potential expansions.
7. **"References"** – Lists the supporting documents and sources referenced in the deliverable.
8. **"Annexes"** – Provides additional details on specific unit testing (*Annex I*) and integration testing (*Annex II*) methodologies and technical validation processes, as well as Intelligence Layer-related updates (*Annex III*) on the final release.

2 ICOS Final Release Notes

The ICOS Final release represents the third official software release within the ICOS project, made available in month 32 of the project timeline (April 2025). It introduces several enhancements compared to the first and second ICOS releases. Notably, the updated versions of both the source code and technical documentation have been made *publicly accessible*.

This section provides a concise overview of the key new features introduced in this release (see *Section 2.1*), along with direct links to online resources, including the source code (*Section 2.2*), software artifacts (*Section 2.3*), and documentation (*Section 2.4*).

2.1 What's New

The final release of ICOS incorporates numerous improvements over the Beta release. Key enhancements include:

- **Refined Unit Testing Framework**: Expanded unit test coverage across components, ensuring greater reliability and stability.
- **New Integration Testbed**: Establishment of a dedicated test environment for end-to-end validation of ICOS deployments.
- **Enhanced Security Measures**: Implementation of advanced security scanning and vulnerability mitigation techniques.
- **Optimized Performance and Scalability**: Improvements in orchestration, monitoring, and resource allocation.
- **New or updated components across ICOS layers**: New features or updated functionalities across different ICOS components.

Regarding the **Meta-Kernel** layer, ClusterLink has been installed on the testbed and has been integrated with the Deployment Management to support applications running across multiple clusters. ClusterLink simplifies connectivity between services hosted across different domains, networks, and cloud infrastructures.

The main new features in the **Aggregator** include:

- Added ClusterLink
- Fixed tag and label display issues
- Added cluster name, pod and node details
- Improved CI/CD, debugging, documentation
- Fixed various errors

As part of Meta-Kernel, **MatchMaker** component has been also enhanced in the final release with the following new functionalities:

- Added node label functionality to allocate specific components to specific nodes as required.
- Multi-component applications deploy across different clusters supporting cross-cluster communication using node labelling.
- Utilisation of the component import-export communication information for more accurate allocation.
- Remediation action to handle a violation by reallocating application components from one node to another.
- Utilisation of forecasted metrics instead of dynamic metrics.

Regarding the **Security layer**, new additions and improvements have been done to improve the security and trust of the ICOS system. In particular:

- Enhancement of the Security Scan with the capability to scan ICOS container images for vulnerabilities (CVE), Infrastructure as Code (IaC) issues and misconfigurations and SBOM (covering host OS packages and software dependencies).
- Adding the User-Managed Access (UMA) protocol to the Identity and Access Manager for controlling the access and authorising the requests to ICOS services.
- Improved the Audit mechanism by using container scanning tool (a Security Scan functionality) as a baseline to produce real time audit logs that focus on identified security aspects from the scanning tool and designing the corresponding observation policies.
- Adding the mutual trust functionality between ICOS components located in ICOS Controller and ICOS Agent, established through a self-signed certificate authority. This also enables encryption of communication between these components using TL.
- Encrypting the communication between ICOS components in the ICOS Kubernetes clusters (in the Controller and in the Agent) using virtual network, namely Container Network Interface (CNI), and Wireguard encryption protocol.

Regarding the **Intelligence layer**, the final release leverages all features reported in D4.2 **[6]**. This includes:

- **AI coordination module**: The ICOS Shell can request training and predictions to the intelligence layer, as well as showing and removing available models. This performed through Keycloak auth into the Intelligence coordination frontend (metrics export API), which interacts with the backend API which owns the ICOS Controller AI model registry.
- **Data processing module**: Improved compatibility with persistent data through data management.
- **AI Analytics module**: Now supports multivariable forecasting, enabling users to predict any combination of the selected metrics. The module also features integrated model training, triggered via the frontend API or ICOS Shell, which dynamically queries Prometheus for relevant data. Additionally, model compression for PyTorch models is now available in the Intelligence API backend, optimizing performance and resource usage.
- **TrustworthyAI module**: Enhanced functionality integrated into the AIOps tools including model explainability, AI model learning curves to support experiments, drift detection and confidence intervals at runtime/operational level, and integrated general-purpose support for Federated Learning (FL).
- **AI Model and Data Catalogue** (formerly AI marketplace): integration between online repositories and backend Intelligence API.

The final release also includes AI support for ICOS users, through the AI support containers. These containers leverage the functionalities of the intelligence layer modules, with reduced computational consumption to operate in edge devices. These containers also include Jupyter Notebooks to allow ICOS users run their own experiments using an identical Python environment for data analysis and AI related purposes.

Regarding the **Data Management layer**, the final release leverages the features presented in D4.2 **[6]**, including:

- *Openness and Interoperability:* ICOS users can leverage the Eclipse Zenoh protocol, which unifies data at rest, data in motion and computations. In the final ICOS release, the Zenoh v1.3.3 (see https://github.com/eclipse-zenoh/zenoh/releases/tag/1.3.3) was released in the staging testbed managed by NCSRD. Zenoh querier API supports efficient and optimised data retrieval.
- *Adaptability and Scalability:* Zenoh v1.3.3 introduces features that adapt to varying technological needs. The stabilization of liveliness API support ensures real-time monitoring of active participants in the network. Added a Transistor-Transistor Logic (TTL) connection parameter to multicast. Zenoh-Pico, which is made for small IoT devices, has also been updated to work with the latest version.

- *Security and Privacy:* Added Access Control (ACL) policy in the config file. The protocol enhancements address critical issues such as fragmentation and message integrity, ensuring secure and reliable data transmission. These updates safeguard interactions across the network, whether between IoT devices or in the cloud.
- *Performance Optimisation:* In addition to the querier, the new advanced publisher/subscriber mechanisms also improve data throughput and reliability. These optimisations enhance system performance while minimising resource consumption, key to efficient IoT-to-cloud integration. Added enforced downsampling configuration, and reduced routing resource tree memory consumption.
- *Technology Agnosticism:* Zenoh's v1.3.3 is aligned with ICOS objective and is committed to supporting diverse hardware and software platforms, including new features for QNX operating systems, highlighting its technology-agnostic approach. This openness reduces dependence on specific vendors and encourages widespread adoption. There is an ongoing effort to integrate natively Zenoh, as Tier-1 communication middleware in the coming ROS2 release in May 2025; this is achieved in rmw_zenoh (see https://github.com/ros2/rmw_zenoh).

Finally, regarding the final web-based **ICOS Shell**, new features are:

- 2FA for backend, and CLI.
- Intelligence layer integration (train, predict, get, delete models).
- Controller selection from the lighthouse upon first login.

Regarding the **ICOS GUI** (Graphical User Interface), the new features include:

- ICOS ecosystem view, includes Controllers, Agents and nodes.
- Deployment Control:
  - New Start Deployment and Stop Deployment actions have been added to the Deployments section.
  - A Deployment Status column has also been included in the deployments list for better visibility.
- Deployment Creation:
  - A new section has been introduced for creating deployments.
  - Users can now select YAML files and initiate deployments directly through the interface.
- Additional Notes:
  - Minor adjustments were made in the graph section, including the removal of the Pod component.
  - A Delete option has been added under Deployments. However, the system currently displays a message indicating that some child jobs are not yet undeployed. This suggests that additional service support may be needed for child jobs.
- Two-Factor Authentication

2.2 Source Code

The code of the **ICOS Final release** is open-source and publicly available in the following GitHub organisation *icos-project* available at https://github.com/icos-project.

The ICOS GitHub organisation is divided into multiple repositories, which are structured as follows:

- ICOS-Meta-OS: Main ICOS system codebase.
- Shell: ICOS Shell tools.
- ICOS Suites: ICOS deployment suites.
- ICOS Documentation: ICOS software documentation.

2.3 ICOS Suites

The ICOS software components developed in WP3 and WP4 are integrated, packaged, and distributed in WP5 as ICOS Suites. This allows for an easier and standardized way of installing and configuring an ICOS Continuum. Compared to the previous ICOS Beta release, the ICOS Suites in the ICOS Final Release have been improved and updated to provide a more uniform, customizable and straightforward installation procedure. To ensure completeness of this deliverable (as the ICOS final release), some of the terminologies, definitions and methodologies are repeated in this document, although they are firstly defined in previous deliverables D5.2[2] and D3.3 [5].

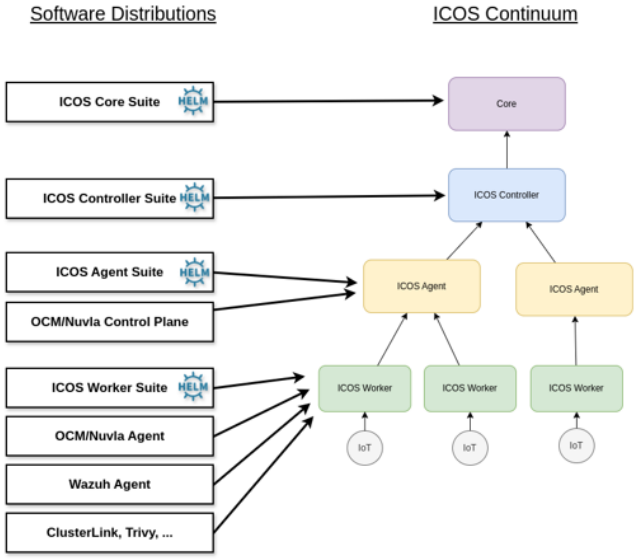

Figure 1. ICOS Suites in the final release.

As depicted in Figure 1, there are four different ICOS Suites, each intended for installing a different type of ICOS node (Core, Controller, Agent, or Worker). The services included in each suite are listed in Table 1. It should be noted that, for some type of nodes, additional software is required or recommended (to enable some specific capabilities). For instance, the orchestrators (OCM or Nuvla) are required to run an ICOS Continuum, but they are not included in the ICOS Suites.

The ICOS Suites are distributed as Helm Charts hosted by the project's artifacts repository at harbor.res.eng.it. The installation follows the usual procedure for Helm. For instance, to install an ICOS Controller:

```
helm install --namespace icos-system icos-controller \
  oci://harbor.res.eng.it/icos/helm/icos-controller \
  --values myvalues.yaml
```

A comprehensive guide and instructions for the preparation, installation, and upgrade of the ICOS Suites can be found in the Developer Guide [7] and the Administration Guide [8].

Table 1. Services in the ICOS Suites.

Suite	Content
ICOS Core Suite	▸ Lighthouse ▸ Identity and Access Management ▸ Certification Authority
ICOS Controller Suite	▸ Job Manager ▸ Aggregator ▸ MatchMaker ▸ Policy Manager ▸ Telemetry Controller ▸ Metrics Exporter ▸ Security Coordinator ▸ Intelligence Coordinator
ICOS Agent Suite	▸ OCM Deployment Manager ▸ Nuvla Deployment Manager ▸ Telemetry Gateway
ICOS Worker Suite	▸ Telemetry Agent ▸ DataClay Metrics Bridge ▸ DataClay

In addition to the ICOS Suites used to deploy an ICOS Continuum, there is also the **ICOS Client Suite** (also known as the ICOS Shell), which includes a set of command-line and graphical tools that users can use to connect to and interact with an ICOS Continuum. The ICOS Client Suite can be downloaded from the public repository at: https://github.com/icos-project/Shell/releases.

2.4 Documentation

GitLab is utilized to store the API definitions (usually in Swagger OpenAPI format) and technical documentation for each component. This centralized and standardized approach ensures easy access for developers who need to integrate with or use the components. By housing all relevant information in one tool and format, GitLab reduces communication overhead and minimizes the time spent locating essential documentation [1]. In the Beta release, we have added new resources as provided in the Section 2.4 of the D5.2 [2]:

- Technical documentation website for ICOS Meta OS is available online at ICOS Documentation (for more details see *Section 3.2*),
- ICOS Project Website provides both technical and non-technical publications, as well as information on Use Cases and project deliverables.
- A collection of scientific publications related to ICOS, outlining research underpinning the ICOS software, is hosted on Zenodo - ICOS Project.

2.4.1 ICOS Project Website

The ICOS Project Website (see Figure 2) serves as a comprehensive resource, offering a wide range of content aimed at both technical and non-technical audiences. It features a variety of publications, including detailed technical papers and accessible non-technical summaries, ensuring that all visitors can find relevant information suited to their background and expertise. In addition, the website includes valuable insights into the project's practical applications through Use Cases, highlighting how the project's findings and technologies can be applied in real-world scenarios. Furthermore, the site provides detailed information on the project's deliverables, offering transparency and updates on the milestones achieved and the outcomes of the ongoing efforts within the project. This makes the ICOS Project Website an essential hub for those seeking to understand the progress and impact of the initiative.

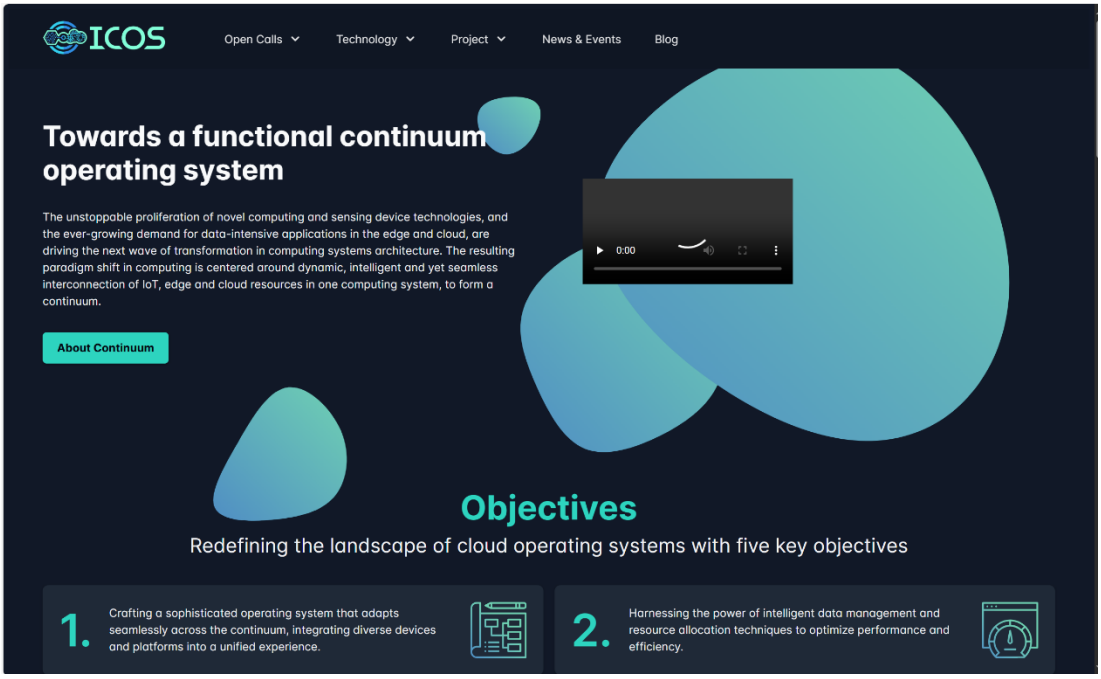

Figure 2. ICOS Project Website

The new website uses the Nuxt.js Framework, a FullStack solution for the Vue.js library. For more information, visit the Nuxt 3 documentation.

The ICOS Project's website has been pre-configured for deployment and is organized into four distinct sections, as outlined in Table 2.

Table 2. The structure of ICOS Website

Section	Description	Naming Convention
blog	Contains articles for the Blog Section.	number.some-name.md (e.g., 20.test.md). The dot after the number ensures sorting from newest to oldest.
news-and-events	Contains news for the News & Events Section.	Same as for the blog.
publications	Contains publications for the Publications Section.	Same as for the blog.
single articles	Rendered when accessed via a specific URL.	No naming restrictions.

2.4.2 Zenodo ICOS Project

A comprehensive collection of scientific publications related to the ICOS MetaOS project –outlining the research and methodologies underpinning the ICOS software – is hosted on *Zenodo* under the ICOS Project repository. This curated collection provides valuable access to research papers, technical reports, and studies that detail the development and advancements of the ICOS MetaOS system. The publications cover a broad spectrum of topics, including the design and architecture of the MetaOS platform, data management strategies, system integration, and the innovative approaches used for carbon observation and monitoring. These resources serve as a vital reference for researchers, developers, and practitioners working on or with the ICOS MetaOS project, offering a deeper understanding of the science and technology behind this critical environmental monitoring initiative.

3 Release Integration

The integration of the ICOS Final release follows the same methodology and policies used for the ICOS Beta release. This process required a coordinated and collaborative effort from all technical partners involved in the project. This section provides

- a brief overview of the work completed for the integration of the latest version of ICOS (Section 3.1),
- the enhancements made to the technical documentation through collaboration and feedback from use cases testing the platform (Section 3.2), and
- the publication of the source code (Section 3.3).

The testing activities conducted during the integration process are discussed in detail in Section 4 and Annexes I and II. Note that, although some methodologies have been already specified in D5.2 [2], they are partly presented also in this document to ensure completeness of the ICOS final release.

3.1 Release Integration Process and Infrastructure

The project has implemented a well-structured CI/CD process, supported by various tools that are set up and maintained by the project partners. This system enables parallel development, integration, and validation activities, significantly reducing the time required for new features to become available for validation. Detailed descriptions of the process and tools can be found in D5.1 [1], with integration tool diagrams provided for reference. For the ease of access, the diagram of the main integration tools (see Figure 3) and the integration process (see Figure 4) are presented below.

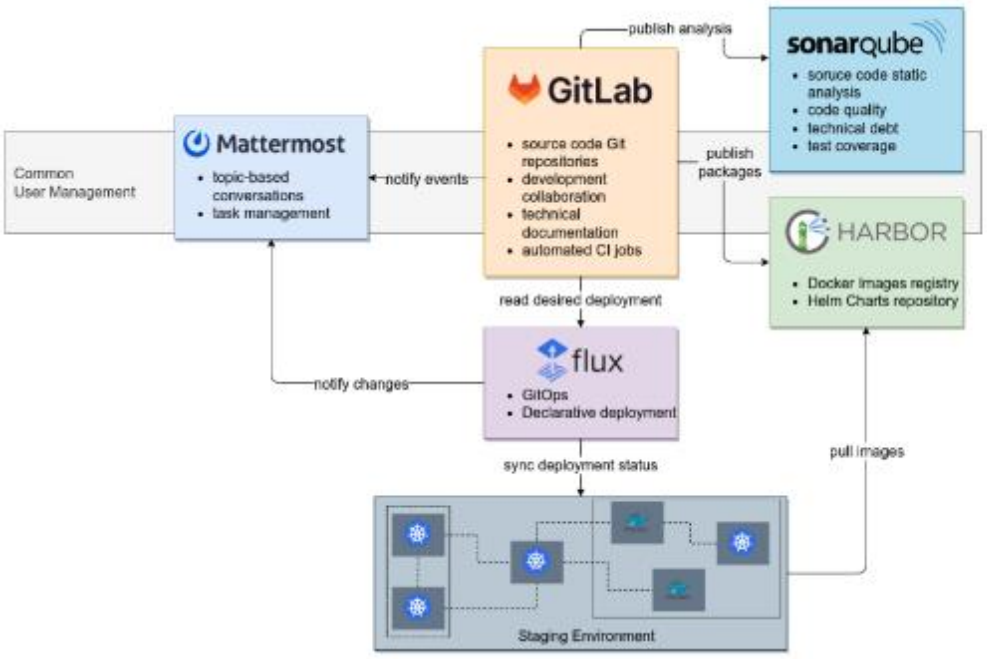

Figure 3. The main integration tools in ICOS [2]

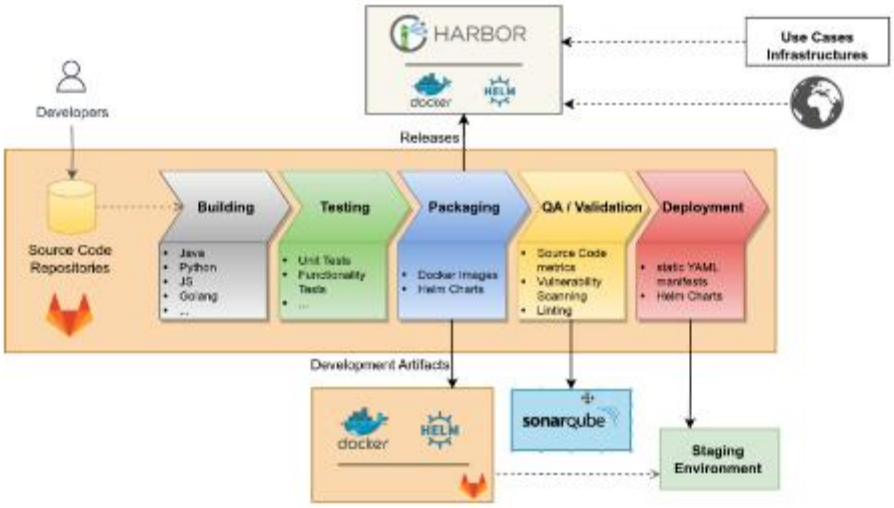

Figure 4. Integration process in ICOS [2]

The integration of the ICOS Final release used the same CI/CD tools and procedures established for the ICOS components. Automated GitLab jobs were employed for building component artifacts (software packages and docker images), checking quality, and deploying them to the staging environment. Additionally, the Infrastructure as Code (IaC) definitions of the ICOS Suites were uploaded to GitLab, following a similar pipeline for integration testing and deployment in the staging environment, all in an automated and seamless manner.

3.2 Technical Documentation

In the ICOS Beta Release [2], we introduced the ICOS Technical Documentation Website. In the ICOS Final Release, we finalized and validated it, introducing the following activities:

- Testbed infrastructures where the editor can contribute to the documentation and the reviewer can test and validate it.
- CI/CD External Repositories were improved.
- Automated the update process in real time by introducing a daemon script.

The structure of the ICOS documentation remains unchanged (more details are provided in D5.2 [2]), and its publication was performed in the same way as before (in manual way).

The **ICOS Technical Documentation** website is stored in the GitLaB ICOS project. The ICOS Technical Documentation **Website** is designed and developed using the Material for MkDocs framework, which provides a clean and modern user interface optimized for technical documentation. This website is based on the concept of writing all its content in Markdown documents. These Markdown files are then processed and compiled into HTML pages, ultimately forming the fully functional website.

One of the key advantages of this approach is its simplicity and flexibility. Since Markdown is a lightweight markup language, it allows authors to focus on writing content without worrying about formatting details. Once the content is ready, MkDocs automatically transforms it into a sleek and responsive site. Additionally, MkDocs introduces specific extensions to Markdown that enhance its functionality, enabling users to leverage additional features such as improved syntax for tables of contents, enhanced navigation, and the inclusion of custom elements like icons and syntax highlighting. For more information on these MkDocs-specific additions, please refer to the ICOS Developer Suites [7].

This approach makes it easy for users to contribute to the documentation. Those who wish to add new content, suggest edits, or create new sections can do so by cloning the documentation repository.

```
> git clone https://production.eng.it/gitlab/icos/documentation/
```

Once cloned, users can simply modify the Markdown files as needed, making changes to the content or structure of the site. Alternatively, users who are managing their own projects can also contribute by adding a **docs** directory to their own project repository. This is especially useful for projects that want to maintain their own documentation in parallel with the main website, while also ensuring that their content is easily accessible and consistent with the broader documentation ecosystem.

By following this simple workflow, contributors can continuously improve the documentation – whether for the ICOS project itself or for their own related projects – while maintaining a streamlined process for content management and version control.

To automate the process of rebuilding the ICOS documentation when changes occur in an external repository, users can configure a CI/CD pipeline. This involves adding a specific CI/CD job in the .gitlab-ci.yml file. The .gitlab-ci.yml file is where you define the pipeline stages, jobs, and rules for automating tasks such as testing, building, and deploying code or content.

```
documentation_website_trigger:
 trigger:
  project: 'icos/documentation'
  branch: 'develop'
 rules:
  - if: $CI_COMMIT_BRANCH == "main"
    changes:
     - docs/**/*
```

The CI/CD job in .gitlab-ci.yml will specify the necessary steps to fetch the updated documentation, run the build process, and regenerate the website with the latest content. Additionally, it may include rules for when and how often the rebuild process should occur, ensuring that changes are reflected on the documentation site as quickly as possible.

To import documentation files from an external repository into your website during the build process, you need to modify the mkdocs.yml configuration file. In this file, specify the external repositories and the paths to the documentation files you want to include:

```
plugins:
- multirepo:
   repos:
   - section: <Section Name>
     section_path: <Navigation Path>
     import_url: <git_repository_url>?branch=<branch_name>
```

An internal wiki is available for editors to help them learn how to navigate and make the most of the documentation system. By providing this internal wiki, the ICOS project fosters a collaborative environment where contributors can easily get up to speed with the documentation process and ensure that the content remains accurate, consistent, and up to date. It is hosted within the *GitLab*

Documentation repository, ensuring that all relevant information is centralized and easily accessible for contributors.

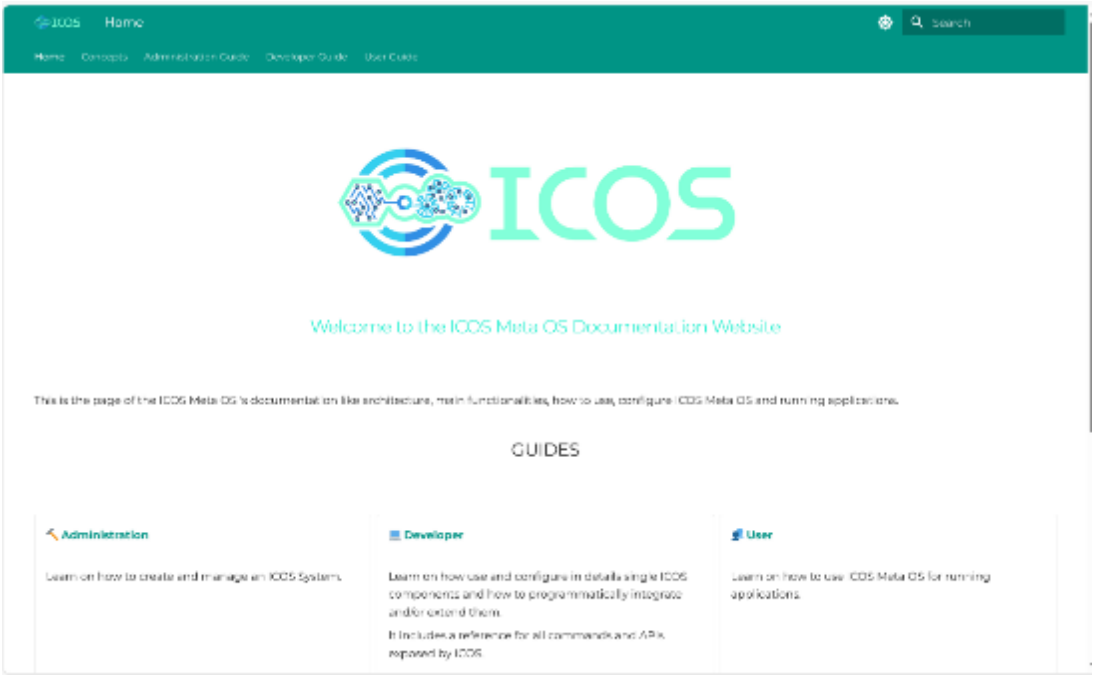

Figure 5. Documentation Site

As described in the D5.2 [2], the structure of the ICOS Technical documentation website (see Figure 5) is provided in the **docs** directory and is divided into four main subdirectories, as shown in Table 3.

Table 3. Subdirectory structure of the ICOS docs directory

Subdirectory	Description
Concepts Guide	This provides a theoretical introduction to ICOS Meta OS, covering its architecture, key functionalities, and components. It is designed for newcomers to ICOS, helping them understand what the system is and how it operates.
Administration Guide	This guide offers step-by-step instructions for setting up and managing an ICOS system. It is aimed at administrators responsible for managing the entire ICOS system or specific components, such as an ICOS Agent.
User Guide	This guide contains tutorials for using ICOS Meta OS to run user applications. Targeted at application developers and integrators, it explains how to use ICOS for application orchestration.
Developer Guide	This focuses on how to configure and integrate individual ICOS components. It includes detailed information on commands and APIs. The guide is intended for technical experts who want to understand ICOS's internal workings or develop custom integrations and extensions.

The testing and validation of the ICOS documentation has been a truly collaborative effort, involving valuable input from both the Use Cases and Open Calls. The provided comments, suggestions, and feedback have played a crucial role in refining the documentation, and thanks to these contributions, we have been able to significantly improve its overall quality and clarity. This feedback loop has ensured that the documentation not only meets the needs of the ICOS community but also stays relevant and accurate over time.

To facilitate easy access, continuous improvement, and efficient maintenance, we have established a dedicated Testbed environment within the ICOS infrastructure. This testbed serves as a staging area where editors can create, modify, and update documentation without affecting the live website. It offers a flexible platform for documentation development, where real-time changes can be made and reviewed.

In the Testbed, reviewers can test and validate the updated content, ensuring that it meets the necessary quality standards before being published. Updates to the documentation are monitored and applied in real-time, thanks to a daemon that continuously checks for changes every minute. This ensures that the editing process is as streamlined and up-to-date as possible.

Once the content has been thoroughly validated and reviewed, it is ready for publication. However, while updates are applied in real-time in the testbed, manual publication is still required to make these changes live on the ICOS Documentation website. This additional step ensures that all updates go through the proper validation process before becoming publicly available.

The source code for the ICOS Documentation is open and publicly available in the ICOS project GitHub repository. This allows the community to access, contribute, and collaborate on the ongoing development of the documentation, ensuring that it remains transparent, open, and easily maintainable for all users and contributors.

3.3 GitHub publication

The process for the publishing of the ICOS MetaOS is implemented in the ICOS Beta release and described in D5.2 [2]. Following the same approach as the previous ICOS release, initially, the ICOS source code is hosted on a private GitLab instance, set up and maintained by the ICOS consortium. This setup allows for internal collaboration while ensuring the code remains private until it is mature and properly licensed.

However, for the ICOS Final release, it is decided to make the source code *publicly available* in a repository. This decision is made because the source code has reached a level of maturity, testing, and documentation that warranted publication. The benefits of this public release include:

- **Easier collaboration** with external developers (e.g., Open Call partners, EU research projects).
- **Increased visibility** within the open-source community, building trust in ICOS MetaOS.
- **Community building**, allowing users to test, extend, and develop tools and applications based on ICOS.
- **Support for exploitation strategies** for ICOS components.

GitHub is selected as the hosting platform due to its large developer community. The code is hosted under the "ICOS Project" organization on GitHub at https://github.com/icos-project (see Figure 6).

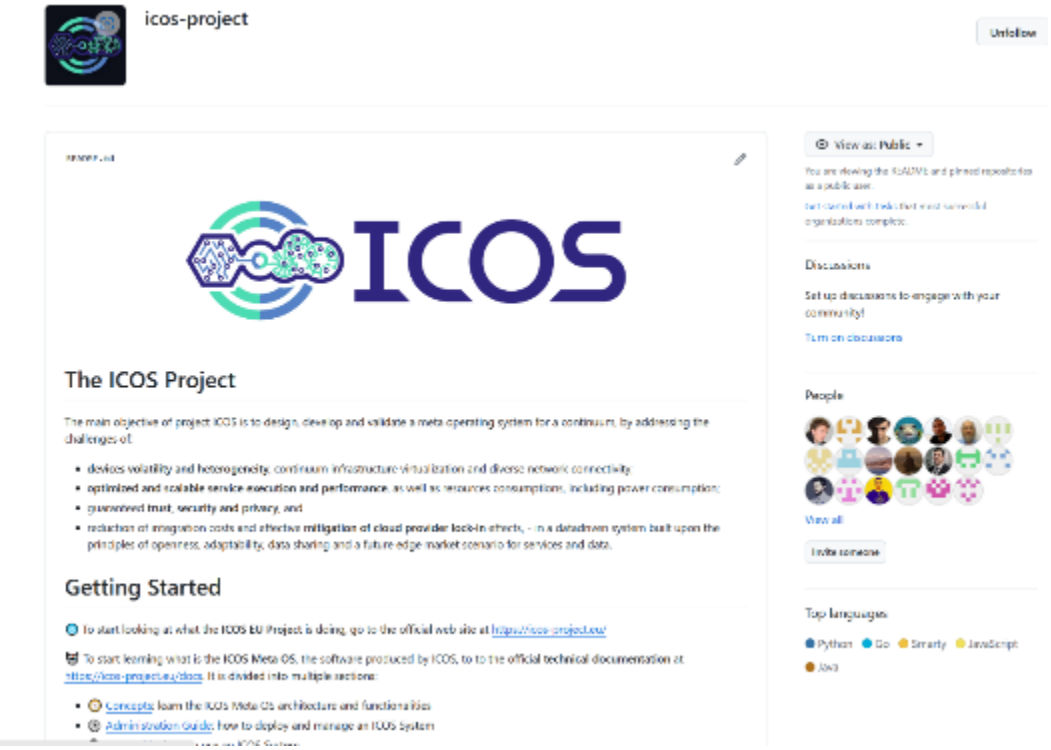

Figure 6. GitHub ICOS MetaOS

The publication process follows two key phases:

1. **Source Code Selection and Preparation:** An internal checklist ensures repositories met publication criteria:
   - Correct open-source licensing and compliance.
   - Presence of README and LICENSE files.
   - Comprehensive documentation (e.g., project description, build instructions).
   - Software quality and security scans, such as SonarQube checks.
   - Automated unit tests (coverage thresholds may be set in future releases).
   - Git tags and release notes for versioning.
2. **Uploading to GitHub:** Since GitHub lacks GitLab's grouping structure, multiple repositories are consolidated into a few top-level repositories using Git submodules. The following repositories were created:
   a. ICOS-Meta-OS: Main ICOS system codebase.
   b. Shell: ICOS Shell tools.
   c. Suites: ICOS deployment suites.
   d. Documentation: ICOS software documentation.

Note that the migration process from GitLab to GitHub is semi-automated to ensure ease of replication in future releases.

4 Release Testing

In the previous deliverable D5.2 [2], *Section 4* provided a comprehensive overview of the testing strategy adopted for the ICOS Beta release. It introduced a rigorous framework for component-level unit testing and Helm testing, with dedicated test suites validating the behaviour of key system components such as the Aggregator, Deployment Manager, Intelligence Coordinator, Policy Manager, and others. These efforts were supported by integrated CI/CD pipelines leveraging GitLab and SonarQube to monitor code quality and enforce automated validation workflows. In addition to unit-level validation, D5.2 described the deployment of a staging testbed where functional evaluations were conducted to simulate realistic ICOS environments. The introduction of security scanning, container vulnerability checks, and quality assurance mechanisms further strengthened the robustness and trustworthiness of the ICOS release.

Building on this foundation, *Section 4* of the present deliverable continues this trajectory by reporting the expansion of unit tests, both through the creation of new test cases and updates to existing ones. Furthermore, this release marks the execution of the first integration tests, enabled by a newly established dedicated testbed infrastructure, which simulates real-world conditions for full system interaction testing. These integration efforts validate the overall cohesion, interoperability, and scalability of ICOS in complex deployment scenarios, reflecting its transition from a beta-grade system to a production-ready platform.

Additional details on testing are provided in the annexes at the end of the document:

1. **Annex I – Testing Activities**: This section outlines the various types of tests conducted to cover critical areas of the application lifecycle, including component connectivity, functionality verification, configuration accuracy, integration readiness, and operational behaviour.
2. **Annex II – Integration Testing Methodology**: This part elaborates on the systematic approach taken for integration testing, detailing key elements such as test cases, test description, preconditions, test steps, expected results, actual results, and status. The methodology ensures that each mandatory unit is tested in integrated test scenarios for validating system-wide interactions.

By structuring the section in this manner, the objective is to provide a comprehensive and coherent narrative of the testing strategies and methodologies employed in the ICOS project, highlighting the commitment to delivering high-quality and reliable software solutions.

In addition to the testing strategies inherited from D5.2 [2], this section also draws upon insights from deliverable D3.3 [5], which documents the final integration of the Meta-Kernel Layer. D3.3 [5] introduced advanced testing efforts for components such as ClusterLink, Topology Exporter, and the Distributed and Parallel Execution module, emphasizing both unit-level validations and complete integration loops. Notably, ClusterLink underwent a comprehensive testing campaign that included code formatting/linting, unit tests, and end-to-end (E2E) scenarios involving inter-cluster communication and access policy enforcement. These efforts ensured its reliability as a backbone for cross-cluster service connectivity.

The Distributed and Parallel Execution (D&PE) component, on the other hand, was subjected to new tests verifying its dynamic response to topology change notifications sent via Zenoh messaging, showcasing runtime adaptability. Similarly, the Topology Exporter was validated against mocked Aggregator data to ensure correct propagation of deployment changes back to subscribing application components.

This broader test spectrum not only supports the validation of individual modules but also strengthens the system's interoperability, scalability, and observability across multiple layers of the ICOS stack.

These additional testing activities are also reflected in *Annex I*, where a comprehensive summary of unit tests and integration tests introduced in both D5.2 [2] and D3.3 [5] is provided for clarity and completeness.

4.1 Methodology and Implementation

4.1.1 Unit Testing and Helm Testing

The testing strategy adopted for the ICOS Final Release builds upon the comprehensive foundation laid during the ICOS Beta Release, as detailed in *Section 4.1* of D5.2 [2]. That section described the systematic implementation of unit testing and Helm testing across ICOS components. These tests covered critical modules such as the Aggregator, Deployment Manager, Intelligence Layer, Job Manager, Policy Manager, Scaphandre, and Security Layer. Methodology and structure of those tests, including the template and classification of test cases, are fully documented in *Annex II* of D5.2 [2].

In the current final release, additional testing efforts were made to expand the coverage and robustness of the system:

- Several **new Unit Tests** have been implemented across different components. Some existing test suites have also been revised and extended to reflect updates in component behaviour and interdependencies. Note that, in this document, we simply report the definition of the unit tests that were performed offline. Execution of the tests followed formal procedures ensuring bug-free installation and functional stability of the deployed components.
- Among the newly introduced test cases, the **Matchmaker** component, **Metrics Export API** component (Intelligence Coordination Frontend as presented in D4.2 [6]), **Policy Manager** and **Federated Learning** component have been explicitly covered. These tests validate the component's capacity to allocate deployments to suitable nodes based on application requirements and infrastructure metadata.

To preserve structural consistency with previous deliverables, test definitions for the new components will be presented in tabular format, aligned with the tables in *Section 4.1* in D5.2 [2]. Each table contains test definition, interfaces under test, and a brief description of the test cases.

Implementation

1. **Matchmaker** (seeTable 4) - The component tests for the Matchmaker are designed to ensure that it can correctly receive input files (application descriptor and ICOS topology), process the matchmaking logic, and return the appropriate target nodes for each application component.

Table 4. Matchmaker Test Suite

Definition of the Unit Tests	Verify the deployment of the Matchmaker component in a Kubernetes cluster. Ensure it processes the application YAML file and ICOS topology JSON file, applies the matchmaking logic, and returns the best possible target node for each component of the application.
Definition of the Interfaces to be tested	The Matchmaking API receives the application YAML file from the Job Manager and the ICOS topology file from the Aggregator.
Unit Test Cases descriptions	**UT1:** Successful deployment of Matchmaker component on a Kubernetes cluster. **UT2:** Assign a target node for a single-component application with no requirements. **UT3:** Assign target nodes for a multi-component application with specific requirements (e.g., camera device). **UT4:** Assign target nodes for application requiring cross-cluster communication (ClusterLink) using node label functionality. **UT5:** Return an error when no suitable targets are found due to strict deployment constraints.

2. **Metrics Export API** (also called **Intelligence Coordination Frontend**, see **Table 5**) - The component tests for the Metrics Export module are designed to ensure that all its endpoints function correctly and expose, create, update, train, stop, unregister and show metrics appropriately. The Intelligence Coordination Frontend has been documented in https://www.icos-project.eu/docs/Developer/Components/metrics-export/.

Table 5. Metrics Export Test Suite

Definition of the Unit Tests	Ensure that each available API endpoint of the Metrics Export service correctly receives input via JSON payloads and returns valid metric registration, updates, or telemetry-based predictions.
Definition of the Interfaces to be tested	Metrics Export REST API endpoints: • /metrics, • /create_metric, • /unregister_metric, • /create_model_metric, • /stop_model_metrics, • /train_model_metric, • /show_models, • /delete_model.
Unit Test Cases descriptions	**UT1:** Validate /metrics returns Prometheus-compatible format. **UT2–UT5:** Test creation of each supported metric type (Counter, Gauge, Info, Enum). **UT6:** Test unregistering a metric. **UT7:** Test model-based metric creation. **UT8:** Test stopping metric collection. **UT9:** Trigger model training with telemetry input. **UT10:** Retrieve available models from the registry. **UT11:** Delete a model from model registry

3. **Federated Learning** (see **Table 6**) – Federated Learning component has been reported in detail in D4.2 [6]. Test cases for this component have been divided into 3 classes, namely Fetcher tests (class 'UT-F', UT-F1 to UT-F5), LSTM tests (class 'UT-L', UT-L1 to UT-L6), and Processor tests (class 'UT-P', UT-P1 to IT-P6). ICOS FL component has been documented in https://www.icos-project.eu/docs/Developer/Components/icos-fl/.

Table 6. Federated Learning Test Suite

Definition of the Unit Tests	The ICOS-FL framework tests focus on three main components: Fetcher (for retrieving time series data), LSTM (for predictive modeling), and Processor (for data preprocessing). Tests verify initialization, functionality, error handling, and integration of these components.
Definition of the Interfaces to be tested	The interfaces include class constructors, data retrieval methods, data processing functions, model training operations, and dataset creation utilities. These interfaces accept various parameters including configuration settings, DataFrames, model instances, and return objects like processed data, trained models, and evaluation metrics.
Unit Test Cases descriptions	**UT-F1:** Test initializing a Fetcher instance and validating its configuration. **UT-F2:** Test the TimeSeriesData.get_dataframe method. **UT-F3:** Test the _post_process method for transforming raw data. **UT-F4:** Test the fetch_data method for retrieving and processing time series data. **UT-F5:** Test the _disconnect method for cleaning up resources. **UT-L1:** Initialize LSTM model with default output size. Initialize LSTM model with custom parameters and output size. **UT-L2:** Test forward pass through LSTM model with sample data. **UT-L3:** Test extracting model weights as NumPy arrays. **UT-L4:** Test setting model weights from NumPy arrays. **UT-L5:** Train LSTM model on synthetic data. **UT-L6:** Evaluate LSTM model on test data. **UT-P1:** Initialize Processor with default parameters. Initialize Processor with custom parameters. **UT-P2:** Test data normalization functionality. **UT-P3:** Test train/test splitting functionality. **UT-P4:** Test TimeSeriesDataset class initialization. **UT-P5:** Test retrieving sequence and target from TimeSeriesDataset. **UT-P6:** Test complete data loader creation pipeline.

4. **Policy Manager** (see **Table 7**) – Based on D3.3 **[5]**, this component manages the policies (technical and business performance), detects and predicts violations of such policies in the running application.

Table 7. Policy Manager Test Suite

Definition of the Unit Tests	Verify the deployment of the Policy Manager, ensure all the components are functioning correctly, and confirm the User Interface (UI) is accessible through a web browser.
Definition of the Interfaces to be tested	Policy Manager Backend Service and Web UI.
Unit Test Cases descriptions	**UT1**: Successfully launching and running the Helm release for the Policy Manager. **UT2**: Run python unit tests (47 total test cases). **UT3**: Access the Policy Manager UI. **UT4**: List of all Policies created is showed from the Policy Manager UI. **UT5**: History of the policy selected is showed from the Policy Manager UI. **UT6:** Edit a policy variables and test that the policy evaluation expression changed accordingly **UT7:** Trigger a policy violation through the debugging API and check that remediation action is triggered by the Policy Manager

5. Note that, unit tests for **ClusterLink** are based on the Go testing framework, enabling fast checks for specific units such as the ClusterLink operator, ClusterLink control plane controllers (authorization, XDS, CRDs), ClusterLink dataplane controllers, the policy agent, and various utilities used by ClusterLink components. Additionally, there are E2E tests to verify the correctness of the system. This framework uses Kind and the testify/suite package, a testing library for Go, to test different end-to-end scenarios, such as CLI commands, ClusterLink operators, correct usage of CRDs, proper implementation of the policy agent for access control rules, and various load balancing scenarios. These tests can be found in ClusterLink test suite.

Execution and Results

Additional testing activities and results, including per-component Helm charts and CI/CD pipeline test runs, remain consistent with the approaches adopted in D5.2, confirming the stability and repeatability of all component-level operations.

By combining extended unit tests with a fully functional integration testing framework, the ICOS Final Release ensures a higher level of software maturity, system coherence, and end-to-end operational integrity across all deployment scenarios.

4.1.2 Integration Testing

Integration Tests have also been executed in this final release. A newly deployed testbed allowed comprehensive validation of real-world deployment flows using (Bash) scripted interactions via the ICOS CLI (e.g., controller registration, user login, application deployment job create/start/stop/delete operations, and others). Specifically, to complement unit and component-level verification, an integration test scenario has been implemented using a dedicated **Bash script** that orchestrates a series of end-to-end validation steps through the ICOS CLI. This script simulates the entire lifecycle of a deployment, including:

- authentication,
- controller registration,
- health checks,
- job: creation, activation, and teardown.

It interacts with the Keycloak-based authentication service, the shell-backend API, and the Job Manager to ensure all critical backend flows operate as expected in a realistic setup. This integration test plays a central role in confirming that the deployed system functions correctly and coherently across services and components.

Four applications were considered during the integration testing, each one testing different features:

- helloword: testing the deployment of one single component with no requirements.
- ffmpeg: the app deploys two components pinning the target nodes using labels.
- compss: deploys a single-component application publishing app-specific metrics and raises a violation.
- hello-world_clusterlink: deploys two components on two different clusters, requiring a ClusterLink-based cross-cluster communication.

In a nutshell, the established tests are executed as a set of commands in an *automated* manner and verify their results. They consist of the actions/steps reported in Table 8.

Table 8. Steps involved in Applications for Integration Testing

Step	Command	Description	Component involved
1	icos-shell auth login	Authenticate against the system	lighthouse, shell backend, IAM
2	icos-shell get resource	List all connected resources	shell backend, aggregator
3	icos-shell create deployment	Create a deployment	JobManager, deployment manager, verifies compatibility with application descriptor
4	icos-shell get deployment	List all deployments	shell backend, JobManager
5	icos-shell get deployment --id XXX	List the created deployment	shell backend, JobManager
6	icos-shell start deployment --id XXX	Start the created deployment	shell backend, JobManager and deployment manager
7	icos-shell stop deployment --id XXX	Stop the created deployment	shell backend, JobManager and deployment manager
8	icos-shell delete deployment --id XXX	Delete the created deployment	shell backend, JobManager and deployment manager

Note that --id XXX corresponds to the deployment identifier assigned in step 3. A comprehensive description of this scenario, its execution logic, and the rationale behind each test phase can be found in *Annex II* of this document.

4.2 Source Code Quality

As described in the deliverable D5.2 [2], most of the ICOS MetaOS software is released as open-source, with the code made publicly available in a repository. Before publishing, the quality of the code must be ensured, which includes several non-functional properties such as readability, maintainability, reusability, reliability, testability, security, documentation, and portability. High-quality code is crucial for the adoption, evolution, and sustainability of the system, particularly after the project concludes. To maintain consistent quality, the ICOS project measures code quality throughout the development cycle, providing developers with continuous feedback. Additionally, a quality check is conducted before publishing any code to ensure it meets the required standards.

The quality of source code in ICOS is assessed using *SonarQube*, chosen for its ability to evaluate multiple aspects of code quality, customizable thresholds, plugin integration for new tools and languages, seamless CI/CD integration, and its open-source nature. Maintained by Engineering S.p.A., SonarQube instance dedicated for ICOS is integrated with the project's GitLab repositories. To ensure high quality code, several jobs have been implemented (see Figure 7):

1. **Dockerfile and Helm Linting (build_deps_helm, lint-heml)**: this process analyzes Dockerfiles and Helm charts to identify errors and detect poor practices. Once all check pass, the changes are published (push_helm).
2. **Secrets Detection (test)**: Scans code repositories for strings resembling passwords, secrets, or private keys, helping prevent secret leaks and promoting best practices.
3. **SonarqubeAnalysis (sonarqube-analysis):** After each commit, an automated job runs to analyze the code and upload the results to SonarQube. If the code fails to meet the established quality threshold, the job fails and provides debugging information.

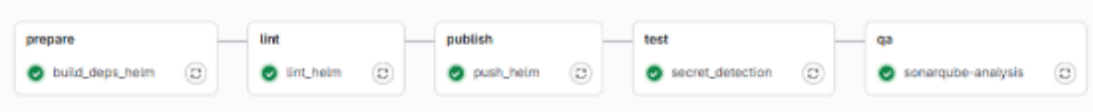

Figure 7. GitLab and SonarQube Integration

As described in the deliverable D5.2, all analyses from GitLab jobs are uploaded to SonarQube, where users can access the current code quality status and historical trends. The portal allows easy aggregation of results across projects and navigation of individual project issues.

4.3 Container Security Vulnerability Scanning

The implementation of the goals and methodology for the code quality and security scanning remains unchanged, as described in the deliverable in D5.2 [2]. Docker images are the main distribution format for ICOS software. Each component of ICOS is packaged into a Docker image, which is built during the CI/CD pipeline. Ensuring the quality and security of the generated images is crucial to guarantee that the delivered software is both reliable and trustworthy.

Multiple tools exist for scanning Docker images. In ICOS, Trivy (https://trivy.dev/latest/) was chosen for its capabilities and ease of integration with the existing GitLab CI/CD pipeline. Trivy can identify various issues in Docker images, including: i) vulnerabilities in OS packages and software dependencies; ii) known Common Vulnerabilities and Exposures (CVEs); iii) IaC issues and misconfigurations; iv) exposure of sensitive information and secrets; and v) software licensing issues.

Trivy is integrated as an automated job in the GitLab CI/CD pipelines for all source code repositories that generate Docker images. The job executes a Trivy scan on the newly created Docker images and reports any detected issues in the execution logs. These scans are systematically performed for all ICOS Docker images. Currently, the results are available only in the execution logs. However, there is potential for improvement by exporting the scan results as metrics, which could then be easily analysed and aggregated (e.g., "total number of high-priority vulnerabilities found").

4.4 Final ICOS Testbed

For the purposes of the ICOS project, NCSRD provides a robust and flexible testbed designed to support the deployment, experimentation, and validation of the ICOS platform. A high-level overview of the testbed is shown in **Fig. 8.** At the core of this infrastructure is a cluster consisting of five high-performance servers. Each server is equipped with **32-core Intel(R) Xeon(R) E5-2640 v3 CPUs @ 2.60GHz**, and has been upgraded to include **125GB of RAM**, offering substantial computing resources suitable for demanding edge computing and orchestration tasks.

For the virtualization infrastructure, **Proxmox Virtual Environment (Proxmox VE)** was used as the core platform. Proxmox VE is an open-source server virtualization management solution that integrates KVM hypervisor and LXC containers, offering a powerful web-based interface and comprehensive CLI tools for managing virtual machines and containers.

To streamline VM deployment and support the dynamic needs of the **ICOS project**, a set of **cloud image templates** were developed and maintained. These templates enabled **rapid provisioning of virtual machines**, significantly reducing setup time and ensuring consistency across deployments. The templates were tailored for various operating systems and pre-configured with essential tools and dependencies required by the ICOS services and testing environments. This approach greatly enhanced the efficiency of the testing and development process, allowing for quick scaling and flexible resource allocation in response to evolving project requirements.

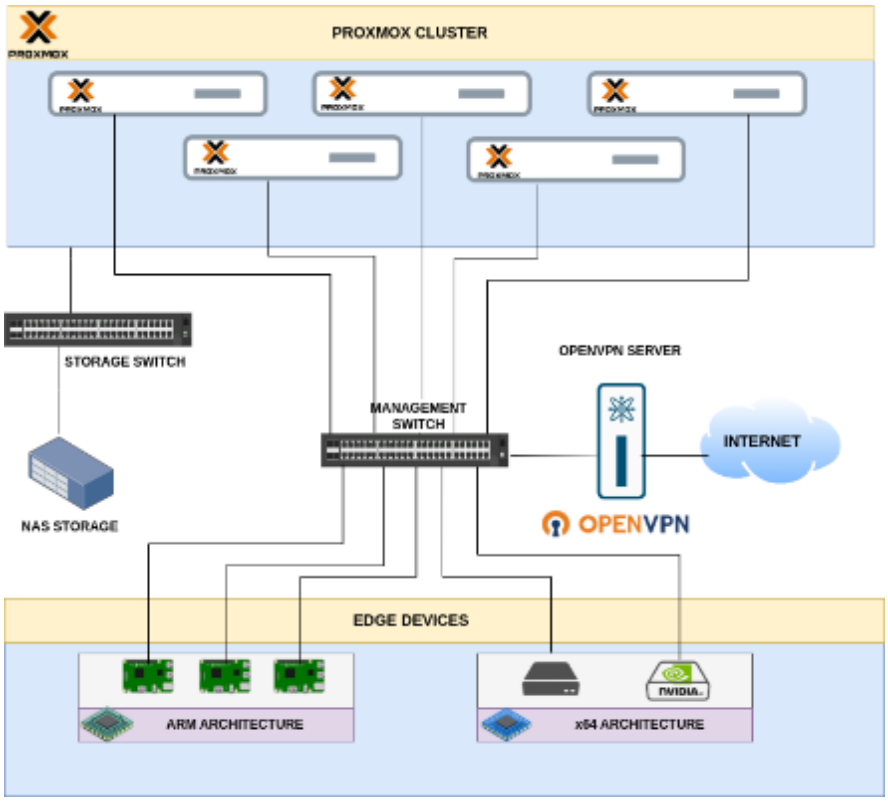

Figure 8. High-level Testbed overview

The system also includes **Network-Attached Storage (NAS)** support to ensure reliable and consistent backup capabilities across the cluster. To facilitate secure and controlled remote access to the infrastructure, **OpenVPN** was deployed and configured. OpenVPN is a widely used open-source VPN solution that provides encrypted connections over the internet, ensuring data confidentiality and integrity. Through this setup, **ICOS consortium partners** were granted seamless and secure access to the internal resources, including testbeds, virtual machines, and development environments. Each partner received personalized credentials and certificates, enabling authenticated access while maintaining strict security policies. This VPN-based access mechanism was critical in supporting the distributed nature of the consortium, enabling collaboration across multiple institutions and geographic locations without compromising the security of the infrastructure.

The cluster is primarily utilized to host the **ICOS Controller and Agent Suites**, serving as the backbone of the ICOS orchestration framework. Additionally, the infrastructure is leveraged to **simulate edge devices**, enabling thorough testing and performance evaluation of ICOS in various distributed and resource-constrained scenarios.

To further enhance the capabilities of the ICOS testbed, **NCSRD has added a GPU-enabled server featuring an NVIDIA AD104GL L4 GPU**, expanding the platform's capacity to support AI/ML workloads and GPU-accelerated processing tasks within ICOS deployments. Specifically for the GPU-enabled edge device, the **NVIDIA device plugin** was deployed to enable seamless integration with Kubernetes workloads. The setup also utilizes **time-slicing support**, allowing multiple pods to share the GPU efficiently. This configuration is implemented on a **K3s distribution**, ensuring lightweight and performant orchestration suited for edge and resource-constrained environments.

In addition, the testbed is **enriched with a range of physical edge devices** supporting both **ARM and x64 architectures**, facilitating **multi-architecture (multiarch) deployments** and validation. This diversity ensures the ICOS platform can be thoroughly tested in heterogeneous environments, closely resembling real-world edge computing scenarios. This setup offers a highly adaptable and secure

environment for the development and validation of ICOS technologies, supporting both centralized and decentralized deployment models in real-world-like conditions.

NCSRD provided and maintained a robust testing infrastructure comprising three distinct testbeds to support the development and validation processes:

1. **Staging Testbed** - Used for preliminary testing of new features and configurations before further evaluation. The topology and the underlying technologies are depicted in Figure 9. The staging testbed consists of a hybrid setup that includes both virtual machines (VMs) and physical devices, All virtual machines in the staging testbed are managed via Proxmox VE and provisioned using pre-configured cloud Ubuntu 22.04 images, ensuring consistency and rapid deployment. These VMs are primarily used for service testing, development, and continuous integration workflows. For the edge devices besides the simulated ones the testbed was enriched with the addition of various physical instances with arm and amd64 architectures.
2. **Integration Testing Testbed** - Dedicated to testing interoperability between components and ensuring seamless integration (see Figure 10). This minimal setup which is purely virtualized allows the testing of the ICOS components.
3. **Stable Testbed** - Served as a reliable environment for validating stable releases and reproducing issues identified in production. The testbed was built to support remote onboarding of both the ICOS Agent and edge devices (see Figure 11). This was important for running the project's use cases and for allowing external teams from the Open Call projects to easily connect and test their solutions.

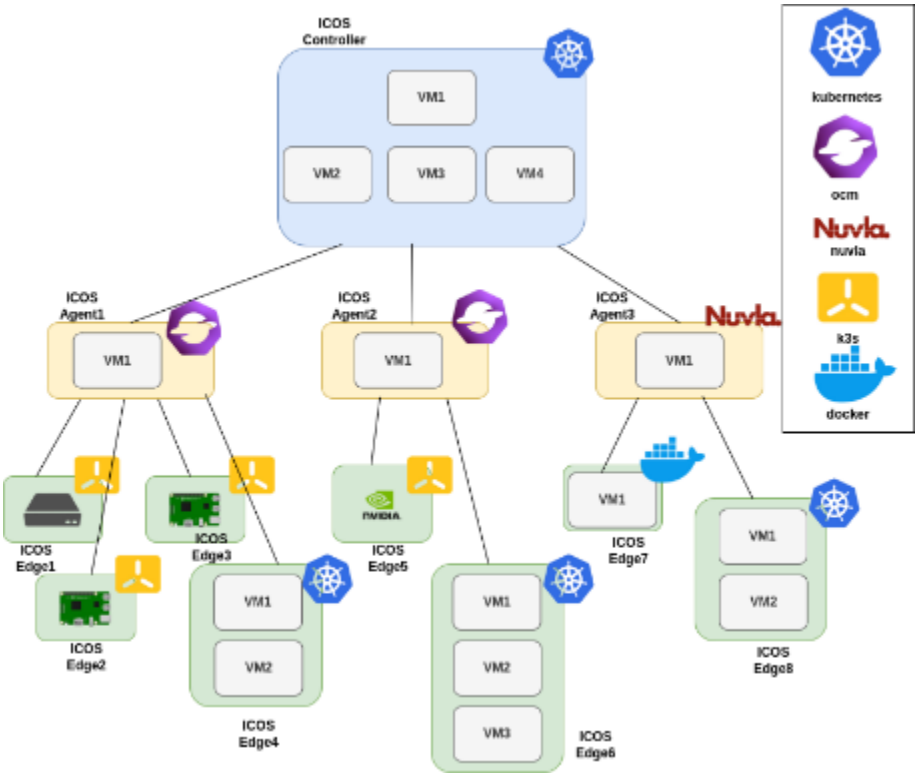

Figure 9. Staging Testbed

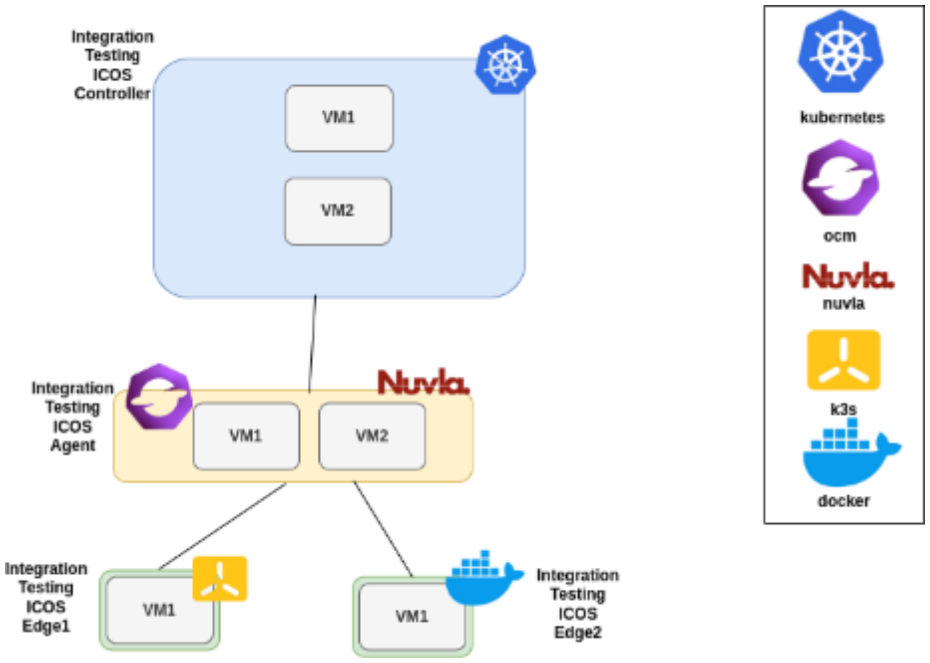

Figure 10. Integration Testing Testbed

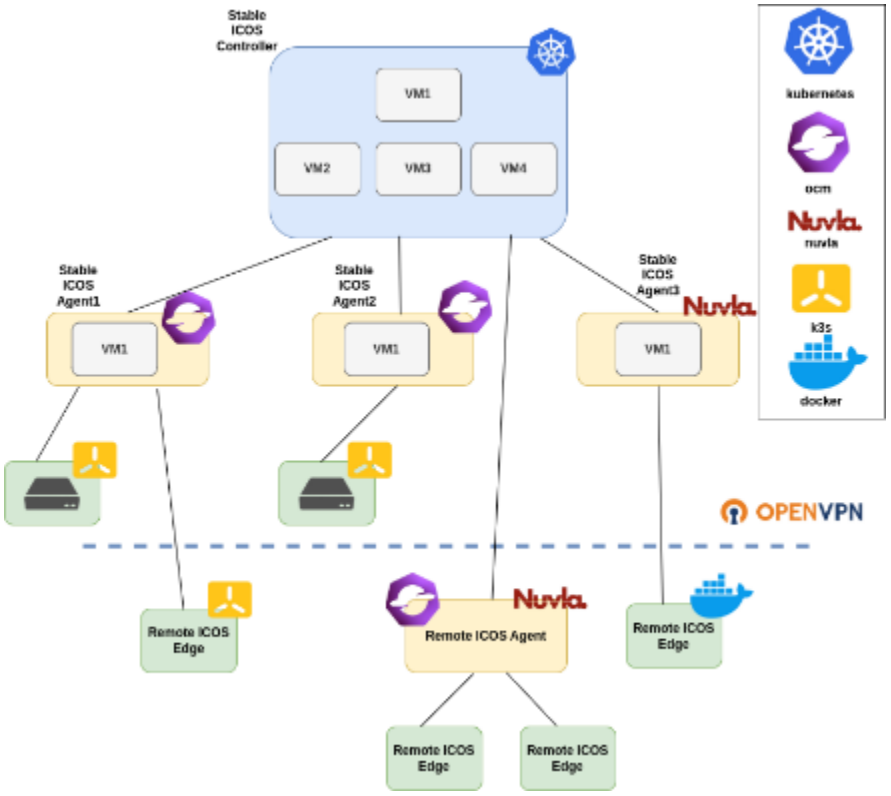

Figure 11. Stable Testbed

Remote onboarding was made possible using **OpenVPN tunnels**, with each user or device assigned a **dedicated account**. This setup allowed partners and contributors to securely connect to the infrastructure from different locations without needing physical access. Once connected, users could:

- Deploy and run the ICOS Agent on supported devices (VMs or physical nodes)
- Onboard edge devices
- Access shared resources like testbeds and services

5 Final Release

The ICOS Final release was the result of collaborative efforts, mainly adding the functionalities and features reported in *Section 2.1* of the present document. In Section 5 of D5.2 [2], an analytical list of the planned functionalities to be added between ICOS Beta and ICOS Final release was provided. Although this list represented only a tentative plan at the moment of D5.2 submission, the project's efforts and the new functionalities achieved in the final release are closely align with the reported items. To maintain consistency across WP5 deliverables, Table 9 summarizes the expected functionalities reported in D5.2 alongside those achieved in the ICOS Final release.

Table 9. Planned vs. achieved functionalities from D5.2 to D5.3.

ICOS domain	Functionality planned in D5.2 [2]	Achieved in D5.3?
Continuum Management	Onboarding and management of devices will be eased and partially automated.	✓
	Deployment of multiple ICOS Controllers that will coordinate through cross-communication protocols.	
	ClusterLink integration for full network connectivity across ICOS sites.	✓
Runtime Management	Definition of the syntax for describing the application topology and orchestration requirements (Application Descriptor).	✓
	Matchmaking process will be based on AI forecasting values to provide better deployment strategies using historical and run-time telemetry and monitoring data.	✓
	Dynamic adaptation loop by (i) publishing new metrics in the monitoring infrastructure (i.e., security and green efficiency related) for better service behaviour insights.	✓
Data Management	Task offloading mechanism will allow the Intelligence Layer to perform decentralized training.	✓
	Data Management service on the ICOS agents will give a direct access to agent data, allowing other components to interact and react in a decentralized manner with lower latency and lower bandwidth requirements.	✓
Intelligence Layer	Execution of mitigation or recovery actions for detected anomalies, security vulnerabilities and threats.	✓
	Enhance the basic management of security compliance policies with the ability to execute remediation actions for compliance policy violations.	✓
	Zero-trust approach will be completed adding authentication, authorization and encryption not only for external communications, but also for intra-node, service-to-service calls.	✓
Shell	Visualization of real-time performance metrics and alerts.	✓

6 Conclusions

This document accompanies the ICOS Final release, which marks the third official software version delivered by the ICOS project. Compared to the previous release (ICOS Beta [2], June 2024), the ICOS Final release concludes a series of enhancements across all ICOS system layers. *Section 2* provides an overview of the newly delivered functionalities and directs readers to relevant online resources associated with this release.

Two major integration activities finalized in the ICOS Final release have significantly boosted the reach, usability, and long-term viability of the ICOS Meta OS:

- The source code developed for the ICOS MetaOS has been updated and made publicly accessible as open-source via icos-project on GitHub.
- Comprehensive technical documentation associated with all the advancements is updated and became available online at ICOS Documentation, aiming at supporting the deployment and management of an ICOS System.

These contributions relied on a strong collaborative effort among the project's partners. The methodologies, processes, and results tied to these efforts are detailed in *Section 3.2* and *Section 3.3* of this document.

Section 4 outlines the project's progress in establishing a formal and systematic software testing framework. It sets out objectives, strategies, and current testing practices. A robust unit and integration testing strategy has been put in place to ensure the reliability and correct functionality of both individual and inter-connected software components within the ICOS ecosystem. This rigorous approach helps maintain a high level of quality, reliability, and maintainability across ICOS components.

Additionally, a set of automated quality assurance tools is integrated into the CI/CD pipeline to enhance code and artifact validation. These include:

- Source code analysis with SonarQube,
- Container vulnerability assessment,
- Detection of embedded secrets,
- Linting checks for Docker and Helm definitions.

Lastly, *Section 5* summarizes the key achievements of the ICOS Final release, comparing the newly added features with those planned in the ICOS Beta roadmap (see Section 5 of D5.2 [2]).

Dissemination activities around ICOS final release are crucial for showcasing results, attracting stakeholders, ensuring impact, and promoting adoption. Here's a list of dissemination activities that are done or planned:

- Public Project Website & Final Release Page with a dedicated section for the final release, including downloadable software artifacts and documentation.
- Demos, videos or white paper showcasing ICOS final release.
- Blog post summarizing lessons learned and future directions.
- Series of posts on Twitter, LinkedIn, Mastodon, etc., highlighting major milestones, key results, success stories from pilots or experiments, final release announcement teaser with visuals or videos.
- Further scientific dissemination with journal papers, conference presentations, posters and demos.

7 References

[1] ICOS. D5.1 - First ICOS Release: ICOS Alpha Release, M. Michalke, 2023. https://icos-project.eu/files/deliverables/D5.1_Alpha_release_v1.0.pdf. Retrieved 29/04/2025

[2] ICOS. D5.2 - ICOS Beta Release (IT-2), G. Giammatteo, 2024. https://icos-project.eu/files/deliverables/D5.2_ICOS_Beta_Release.pdf. Retrieved 29/04/2025

[3] ICOS. D2.3 - ICOS ecosystem: Technologies, requirements and state of the art (IT-2). G. Xylouris, 2024. https://icos-project.eu/files/deliverables/D2.3_ICOS_ecosystem_Technologies_requirements_and_state_of_the_art_v1.0.pdf. Retrieved 29/04/2025

[4] ICOS. D2.4 - ICOS architectural design (IT-2). J. Garcia, 2024. https://icos-project.eu/files/deliverables/D2.4-architectural-design-it2.pdf. Retrieved 29/04/2025

[5] ICOS. D3.3-Meta-Kernel Layer Module Integrated IT-2. K. Meth, 2025. https://icos-project.eu/files/deliverables/D3.3-Meta-Kernel_Layer_Module_Integrated_IT-2.pdf. Retrieved 29/04/2025

[6] ICOS. D4.2 - Data management, Intelligence and Security Layers (IT-2). AL. Suárez-Cetrulo, 2025. https://icos-project.eu/files/deliverables/D4.2_Data_Management_Intelligence_Security_Layers_IT-2.pdf. Retrieved 29/04/2025

[7] ICOS Core Suite - ICOS MetaOS. Developer Guide section. https://www.icos-project.eu/docs/Developer/Suites/Core/ . Retrieved 29/04/2025

[8] ICOS. Administration Guide - ICOS MetaOS Administration Guide section. https://www.icos-project.eu/docs/Administration/gettingstarted/ . Retrieved 29/04/2025

Annex I: Testing activities

This annex presents an overview of the testing activities carried out throughout the ICOS project, with a focus on three key deliverables: D5.2 (Beta Release) [2]**,** D5.3 (Final Release), and D3.3 (Meta-Kernel Layer Integration) [5]. It consolidates the various testing methodologies employed at different stages of the system's evolution, including unit tests, Helm-based validations, integration tests, and system-level evaluations.

1. **Testing Activities from D5.2 –** D5.2 introduced the initial systematic approach to testing within ICOS, focusing on:

    - **Unit tests and Helm chart tests** for components such as the Aggregator, Deployment Manager, Intelligence Layer, Job Manager, Policy Manager, Scaphandre, and Security Layer.
    - Use of **CI/CD pipelines (GitLab)** and **SonarQube** for continuous integration, code quality checks, and vulnerability scanning.
    - A **staging testbed**, simulating realistic ICOS deployments, enabled early validation of component interactions and application lifecycle flows.
    - Testing methodologies were documented in a formal test case structure (test ID, description, expected vs. actual results), as detailed in *Annex II* of D5.2 [2].

2. **Testing Activities in D5.3 –** Building on the D5.2 baseline, D5.3 expanded the testing scope by:

    - **Adding new unit tests** and revising existing ones for multiple components, including the newly introduced Matchmaker, Metrics Export API, Federated Learning modules, etc.
    - Introducing a **new integration test framework**, orchestrated through various scripts. The scripts validate system-wide workflows from CLI login and controller registration to deployment creation, management, and teardown using a realistic ICOS CLI interface.
    - Employing a newly deployed testbed for **end-to-end integration scenarios** simulating real user actions.
    - Test coverage now includes API-level validation, Helm-based deployments, and feedback from CI/CD pipelines, increasing assurance of system stability and coherence.

3. **Supplementary Testing Activities from D3.3 –** D3.3 contributed significantly to the robustness of specific ICOS subcomponents, including:

    - **ClusterLink**: Extensive testing including unit tests, end-to-end evaluations, and policy enforcement validation. Scenarios covered cross-cluster communication, dynamic link creation, and user-access policy logic.
    - **Distributed and Parallel Execution (D&PE)**: Tests verified responsiveness to topology change notifications using **Zenoh messaging**, validating event-driven orchestration capabilities.
    - **Topology Exporter**: Tested using mocked Aggregator payloads to confirm correct propagation of deployment information to consuming components.
    - These tests, while developed under WP3 [5], directly support the final integration goals of WP5 and enhance confidence in multi-cluster operability and resource coordination mechanisms.

Component Unit Testing and Helm Testing

For a comprehensive overview of component unit testing and the goals defined, the reader can refer to *Annex I (Section 8.1)* of D5.2 [2]. Testing for the components that are not covered in *Section 4.1.1* of the present deliverable has been already covered in D5.2.

Annex II: Integration Testing Methodology

In addition to the per-component unit and Helm testing activities already described, a comprehensive integration testing script was designed and employed to verify the end-to-end functionality of the ICOS Shell interface and its interactions with the broader ICOS platform components. This testing is encapsulated in a Bash-based script (*helloworld.sh* in the ICOS Shell project repository) which serves not only as a smoke test but also as a narrative-driven validation of the operational lifecycle that a typical ICOS user might experience. Its goal is to verify the full system stack behaviour using the CLI interface alone, simulating:

- authentication,
- registration,
- application deployment,
- application lifecycle control, and
- final cleanup

all executed against a locally running or containerized ICOS environment.

The script begins by compiling the CLI from its Go source using *go build*, targeting the main.go entry point. This entry initializes the CLI using the Cobra framework and routes the user's input commands such as auth login, add controller, or create deployment to their respective functional implementations, usually located within the cmd and pkg/cli folders of the ICOS Shell project structure. In the config-server.yml the endpoints of the needed components (keyloak, lighthouse, job-manager, aggregator, intelligence-api) are set.

Authentication (test 1) is then tested by issuing a login request to a Keycloak instance - representing the IAM - via *./icos-shell auth login*, using credentials provided in a YAML-based configuration file (config_client.yml). The configuration file must include the Keycloak realm, endpoint URL, and user credentials. If these are absent or invalid, the login request will fail silently or return an HTTP error. The CLI logic reads this configuration through the *viper* library and constructs an OpenAPI-generated client call that exchanges the credentials for a valid access token. This token is printed to stdout and persisted in the configuration file for reuse in subsequent CLI calls. The user at the Keycloak instance is configured during its installation.

Next, the script **registers a new controller with the Lighthouse service (test 2)**, typically hosted at 127.0.0.1 unless otherwise specified. This is done using the *add controller* command. This step is crucial because the IP address of the controller is stored back into the same configuration file and directly influences subsequent interactions with the job-manager and shell-backend components. If the controller's IP is incorrectly set (as may occur in Dockerized or multi-host environments), later steps such as deployment creation may fail due to incorrect routing.

Following controller registration, the script **verifies its successful registration (test 3)** by querying the system with *get controller*. This command also updates the selected controller field in the config file. If multiple tests are executed concurrently or sequentially, this automatic modification of the configuration could lead to unstable behaviour. A commonly suggested workaround is to copy config_client.yml to a temporary file during testing, avoiding persistent side effects.

Healthcheck functionality (test 4) is evaluated by simply invoking the CLI without any subcommand. This triggers a default request to the root of the shell-backend API, expecting a 200 OK response. The result is printed directly to stdout. The Go function behind this is likely located in the CLI's root command implementation and uses the previously injected controller host address and scheme to build the request.

A **sample application is then deployed (test 5)** using *create deployment --file app_descriptor_example.yaml*. This is one of the most intricate parts of the script, as it not only validates the controller's readiness to accept workloads but also tests the functionality of the underlying job-

manager and matchmaking systems. The CLI command here produces structured JSON output to stderr and a numeric HTTP status code to stdout, which must both be captured carefully to extract meaningful test results. The script uses a clever null-byte-based redirection trick to separate these two outputs in Bash, parsing the deployment ID from the JSON and the status code from the stdout stream.

Subsequently, the script **verifies that the application deployment is registered and retrievable (test 7)** using get deployment, both as a list query and via a specific deployment job ID lookup. Polling mechanisms are used to handle asynchronous job scheduling delays. If the job ID becomes retrievable and contains the expected metadata, the deployment is considered active.

Once confirmed, **the deployment is started (test 8)** using *start deployment*, and after a pause, it **is stopped (test 9)** using *stop deployment*. Finally, the test concludes with a **cleanup step using delete deployment (test 10)**, thereby confirming that the ICOS backend can gracefully clean up application resources.

As a final step, the script invokes *get resource* to **ensure the shell-backend aggregator can return current system-wide metrics (test 11)** — such as node availability, memory capacity, and telemetry information — as expected.

This scenario not only exercises a realistic usage path through the ICOS system but also validates the interplay between independently deployed ICOS system components. Its value lies in the combination of automation, visibility (via color-coded output), and flexibility to be adapted for future CI/CD integration. While the helloworld.sh script is not invoked automatically by the ICOS deployment itself, it is intended to be run by developers or automated test systems as a validation step. For this reason, its usage is recommended post-setup and post-deployment to ensure the environment is functional before any production workload is submitted.

To successfully run this test, the developer must ensure that all dependent services — including Keycloak, shell-backend, and job-manager — are up and reachable at the IPs provided in the configuration. The CLI must also be compiled on the host OS and be able to execute correctly within its terminal environment. On Windows, for instance, Git Bash or WSL is necessary to handle .sh script execution and Docker volume mounts must be adapted to platform-specific path formats.

By incorporating this script into the ICOS testing ecosystem, the project not only gains a highly practical integration validation tool but also provides an excellent educational resource for developers and integrators who wish to understand and test ICOS in a realistic yet controlled scenario.

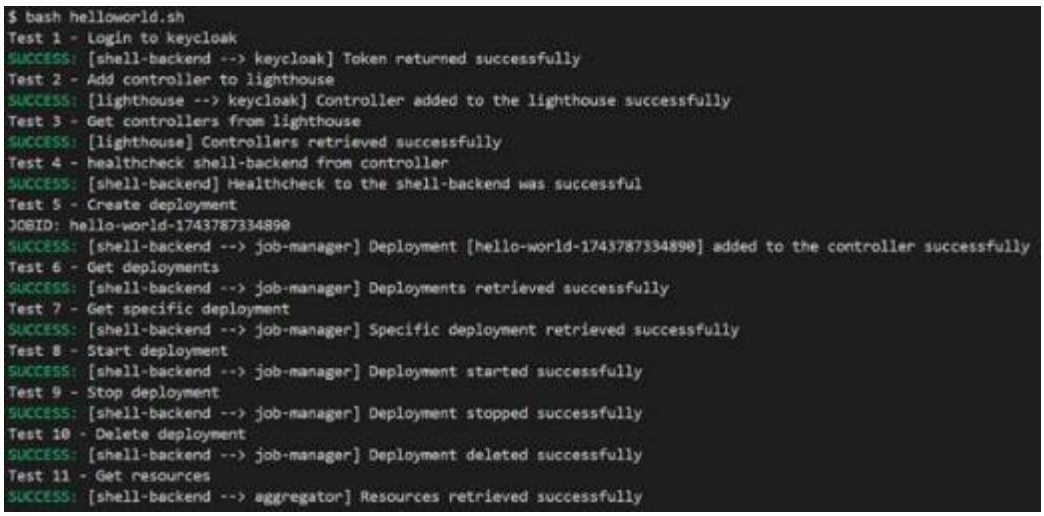

Figure 12. Integration Test Success Run.

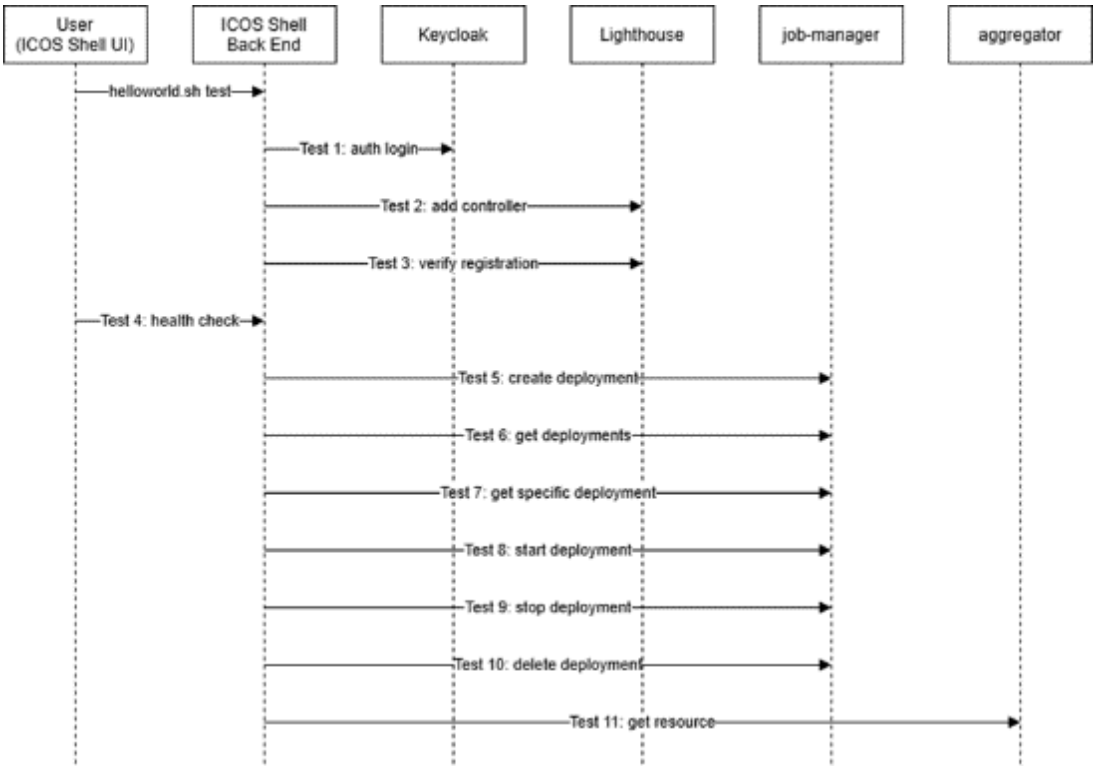

Figure 13. Integration test sequence diagram.

Although four applications were used for ICOS integration testing (see Section 4.1.2), this annex considered indicatively the helloworld application, since it captures the typical interaction patterns and component orchestration within the ICOS platform. Note that the presented methodology is general-purpose and similar procedures were carried out for all integration testing apps.

Annex III - Data Management, Intelligence and Security Layers

Consistent with deliverable D4.2 [6], this appendix provides an update on the status of ongoing activities within WP4.

Regarding Task 4.1 (Data Management), efforts during this reporting period centred on the integration of federated learning capabilities for distributed data access and the validation of GPU compatibility within the dataClay Python library. Furthermore, testing related to data persistency has been successfully concluded.

In Task 4.2 (Intelligence Layer), key integration activities were finalised during this period:

1. The AI Support containers, designed for both edge devices and general ICOS users, have been completed.
2. Integration between the intelligence coordination frontend, backend, and Shell components has been finalised. This includes the implementation of new functionalities for removing metrics and displaying models, as detailed within the present deliverable (D5.3).
3. Federated learning has been integrated as a general-purpose ICOS functionality within the Intelligence API backend. This capability is now accessible via model training API calls and has been fully linked to through the Intelligence Coordination frontend. Additionally, integration through the Swagger API enables the Intelligence Layer to automatically trigger federated training workflows when the corresponding mode is selected. This ensures that federated learning can be initiated and managed directly from the Intelligence Coordination frontend, allowing for a simultaneous orchestration and concurrent operation within the same interface.

In relation to Task 4.3 (Security Layer), work has progressed on identifying suitable underlying ICOS data, specifically Tetragon logs, for the purpose of fine-tuning AI models for anomaly detection. Significant constraints were encountered during the extraction of these logs without imposing excessive load on operating system resources. This challenge has been identified as a limitation associated with Tetragon itself, thereby falling outside the direct scope of the ICOS project. As an alternative mitigation strategy, CeADAR has developed a preliminary model trained on labelled system data designed for system anomaly detection. This model is scheduled for upload to the designated AI Model and Data repository on Hugging Face, as part of the contributions for Deliverable D4.3. It is important to note that this model needs further fine-tuning and requires runtime data flows to be supplied at the operational level for effective deployment.

Tetragon produced relatively static output during application operation, with little variance in the logged events. Furthermore, there was no clear way to distinguish between legitimate actions and potentially malicious behaviour based solely on these logs. This prompted an exploration of system call frequency analysis as an alternative approach. By monitoring the patterns and distributions of system call occurrences over time, a statistical baseline of normal behaviour could be established, enabling the identification of anomalies without relying on predefined signatures or rules.

The attempt to use Tetragon for gathering system call data for ML-based anomaly detection revealed several significant limitations, particularly around resource consumption and data processing flexibility.

When implementing the proof of concept, we observed that monitoring system calls via Tetragon's tracepoint on the sys_enter event created an unsustainable CPU load. Even with rate-limiting policies configured, the sheer volume of syscalls generated by a typical application overwhelmed our collection infrastructure. This high overhead made the approach impractical for production environments, as the monitoring itself would degrade application performance to unacceptable levels.

The core issue stems from Tetragon's inability to implement in-kernel aggregation. Our proposed solution was to maintain a histogram directly in kernel space, incrementing counters for each system

call type rather than sending individual events to userspace. Unfortunately, Tetragon doesn't provide the flexibility to maintain this kind of stateful information within the eBPF program. Instead, each event was forced to be reported individually, creating a data transfer bottleneck. Tetragon has an event reporting mechanism that is optimized for identifying and acting upon individual policy violations, not for bulk statistical analysis. The lack of native support for histogram-style data collection means that implementing our approach would require significant architectural changes to Tetragon itself.

The proposed solution would have been much more efficient if the creation of an array was possible to track frequency of each system call type, increment corresponding counters in kernel space when calls occur, and periodically report only the aggregated histogram data. This would have dramatically reduced the communication overhead between kernel and userspace components. However, Tetragon's current implementation doesn't support this pattern of data collection and aggregation.

Expanding upon Tetragon limitations, there are certain constraints of the LogBERT methodology itself regarding identification and collection of suitable data (logs). The preliminary mode uses the LogBERT methodology that is also used in the Anomaly detection component (LOMOS). Testing of this methodology in several other projects beside ICOS (SUNRISE (HE-101073821) and CYCLOMED (HE-101095542)) has shown that the (self-supervised) training process is sensitive to the proper structure and content of logs as is the case with Tetragon. With LOMOS we tap into logs and data that are typically discarded; however, this gives the raw insight into the application and the system. The logs that the application provides are from the developers themselves, thus offering direct visibility into the application's state. From this, and given the method employed, it follows that the logs must be human-readable and include precise timestamp, i.e., must not be preprocessed in any way. Consequently, not all logs can be used directly.

Regarding content, logs from a system that are guaranteed to be anomaly-free (which can be difficult to produce and verify) should be used to train the model, in order to capture the normal operation or state of the application. Anomalies can then be deliverately introduced to test whether the model can detect them. If the logs already have the anomalies, or if there is no clear way to distinguish between legitimate actions and potentially malicious behaviour (as is the case with Tetragon logs), these anomalies will be incorporated in the training process as a part of normal system behaviour and will not be detected during inference. In such cases, some manual cleaning work must be done to sort out logs that are anomaly-free and then select the anomaly that could be detected. The process is often iterative and requires effort and time.